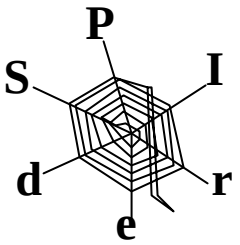
SPIder wordt gesponsord door:

Atos Origin IQUIP CMG Information Technology Quint Wellington Redwood invantive

SPIder Koerier

Januari 2002 www.st-spider.nl



Redactioneel

De donkere, korte winterdagen. Met het uitbrengen van deze koerier is het jaar 2001 al weer voorbij. En zoals altijd hebben we kunnen ervaren hoe snel alles gaat. De overgang van de zwembroek naar de winterwanten is bijna niet merkbaar. De SPIder Koerier zal echter iedereen weer dicht bij het vuur houden van alle SPI-gerelateerde wetenswaardigheden. In deze Koerier een uitgebreid artikel van Niels Malotaux, waarin hij de Evolutionairy Development Methods beschrijft. Een artikel van Erik van Veenendaal over certificatie van Software Testers en diverse nieuwberichten en evenementen. Wij wijzen ook graag iedereen erop dat de SPIder website in een nieuw jasje is gestoken!

Met mijmerende gedachten over het afgelopen jaar en een licht sentiment bij deze laatste in 2001 vervaardigde Koerier, wensen we iedereen een heel gezond en succesvol 2002 toe!

De eerstvolgende SPIder Koerier zal half mei 2002 verschijnen. Uw kopij hiervoor is welkom tot en met 14 april 2002. Voor advertenties en aanmelding van evenementen voor de agendarubriek kunt u contact opnemen met de redactie (redactie@st-spider.nl).

Inhoudsopgave

Evolutionary Development Methods

Niels Malotaux

Software developers systematically fail to manage projects within the constraints of cost, schedule, functionality and quality. Solutions have been developed during the past 35 years, with important results published already some 15 years ago. Still, in practice not much has changed. The challenge is to find ways to catch the practical essence of solutions and ways to get the developers to use these solutions.

In this paper, we show methods and techniques, which do enable software developers and management to successfully managing projects within the constraints of cost, schedule, functionality and quality. These methods are taught and coached in actual development projects with remarkable results.

While software development results were usually delivered late, the delays in other disciplines (like hardware and mechanical development) seemed to be non-existent. Now that we have taught Software Development to deliver Quality On Time (the right things at the right time and within budget), the delays in the other disciplines become exposed. The methods and techniques described in this paper are obviously not limited to just software development. For those projects where delivering Quality On Time is important it is about time that we are going to apply the techniques at the Systems Development level. Therefore the next target for Evolutionary Development Methods will be Systems Engineering.

Introduction

Software developers systematically fail to manage projects within the constraints of cost, schedule, functionality and quality. More than half of ICT users still is not content

with the performance of ICT suppliers [Ernst&Young, 2001]. This is known for some 35 years. Solutions have been developed during the past 35 years, with important results published already years ago (e.g. Mills, 1971 [1], Brooks, 1987 [2], Gilb, 1988 [3]). Still, in practice not much has changed. An important step in solving this problem is to accept that *if developers failed to improve their habits*, in spite of the methods presented in the past, *there apparently are psychological barriers in humans, preventing adoption of these methods*. The challenge is to find ways to catch the *practical essence of the solutions* to manage projects within the constraints of cost, schedule, functionality and quality and *ways* to get the developers to use these solutions.

The importance of solving the problem is mainly economical:

- Systematically delivering software development results within the constraints of cost, schedule, functionality and quality saves unproductive work, both by the developers and the users (note Crosby, 1996: the Price Of Non-Conformance [4]).
- Prevention of unproductive work eases the shortage of IT personnel.
- Enhancing the quality level of software developments yields a competitive edge.
- Being successful eases the stress on IT personnel, with positive health effects as well as positive productivity effects.

In this work, named “Evo” (from Evolutionary), we show methods and techniques which enable software developers and management to deliver “Quality On Time”, which is short for successfully managing projects within the constraints of cost, schedule, functionality and quality. These methods are taught and coached in actual development projects with remarkable results.

The paper is based on practical experiences and on software process improvement research and development and especially influenced by Tom Gilb (1988 [3], later manuscripts [5] and discussions).

History

Most descriptions of development processes are based on the Waterfall model, where all stages of development follow each other (Figure 1). Requirements must be fixed at the start and at the end we get a Big Bang delivery. In practice hardly anybody really follows this model, although in reporting to management, practice is bent into this model. Management usually expects this simple model, and most development procedures describe it as mandatory. This causes a lot of mis-communication and wastes a lot of energy. Early descriptions of Evolutionary delivery, then called Incremental delivery, are described by Harlan Mills in 1971 [1] and F.P. Brooks in his famous "No silver bullet" article in 1987 [2]. Incremental delivery is also used in Cleanroom Software Engineering [6]. A practical elaboration of Evolutionary development theory is written by Tom Gilb in his book Principles of

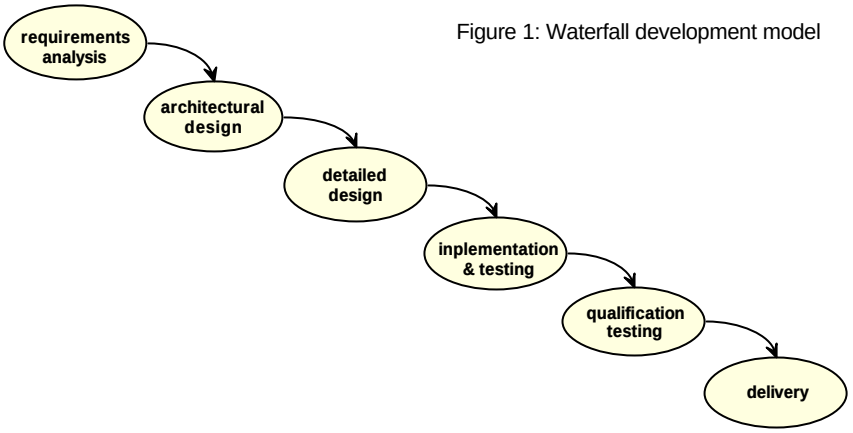


Figure 1: Waterfall development model

Software Engineering Management in 1988 [3] and in newer manuscripts on Tom Gilb’s web-site: http://www.result-planning.com.

Incremental delivery is also part of eXtreme Programming (XP), see http://www.extremeprogramming.org, however, if people claim to follow XP, we hardly see the Evo element practiced as described here. We prefer using the expression Evolutionary delivery, or Evo, as proposed by Tom Gilb, because not all Incremental delivery is Evolutionary. Incremental delivery methods use cycles, where in each cycle part of the design and implementation is done. In practice this still leads to Big Bang delivery, with a lot of debugging at the end. We would like to reserve the term Evolutionary for a special kind of Incremental delivery, where we address issues like:

- Solving the requirements paradox.
- Rapid feedback of estimation and results impacts.
- Most important issues first.
- Highest risks first.
- Most educational or supporting issues for the development first.
- Synchronising with other developments (e.g. hardware development).
- Dedicated experiments for requirements clarification, before elaboration is done.
- Every cycle delivers a useful, completed, working, functional product.
- At the fatal end day of a project we should rather have 80% of the (most important) features 100% done, than 100% of all features 80% done. In the first case, the customer has choice to put the product on the market or to add some more bells and whistles. In the latter case, the customer has no choice but to wait and grumble.

In Evolutionary delivery, we follow the waterfall model (Figure 1) repeatedly in very short cycles (Figure 2).

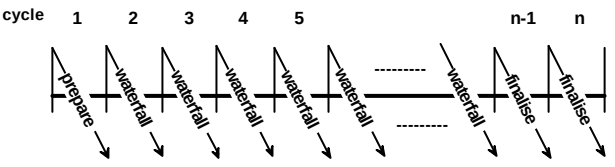


Figure 2: Evolutionary delivery uses many waterfalls

Issues Addressed by Evo

Requirements Paradoxes

The **1st Requirements Paradox** is:

- Requirements must be stable for reliable results.
- However, the requirements always change.

Even if you did your utmost best to get complete and stable requirements, they will change. Not only because your customers change their mind when they see emerging results from the developments. Also the developers themselves will get new insights, new ideas about what the requirements should really be. So, requirements change is a *known risk*. Better than ignoring the requirements paradox, use a development process that is designed to cope with it: Evolutionary delivery.

Evo uses rapid feedback by stakeholder response to verify and adjust the requirements to what the stakeholders really need most. Between cycles there is a short time slot where stakeholders input is allowed *and requested* to reprioritise the list.

This is due to the **2nd Requirements Paradox**:

- We don't want requirements to change.
- However, because requirements change now is a known risk, we try to provoke requirements change as soon as possible

We solve the requirements paradoxes by creating stable requirements *during* a development cycle, while explicitly reconsidering the requirements *between* cycles.

Very short cycles

Actually, few people take planned dates seriously. As long as the end date of a project is far in the future (Figure 3), we don't feel any pressure and work leisurely, discuss interesting things, meet, drink coffee, ... (How many days before your last exam did you really start working...?).

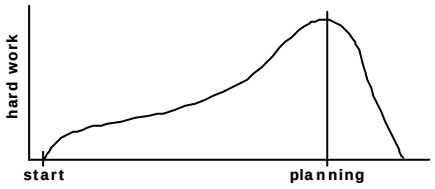


Figure 3: We only start working harder when the pressure of the delivery date is near. Usually we are late.

So at the start of the project we work relatively slowly. When the pressure of the finish date becomes tangible, we start working harder, stressing a bit, making errors causing delays, causing even more stress. The result: we do not finish in time. We know all the excuses, which caused us to be late. It's never our own fault. This is not wrong or right. It's human psychology. That is how we function. So don't ignore it. Accept it and then think what to do with it.

Smart project managers tell their team an earlier date (Figure 4). If they do this cleverly, the result may be just in time for the real date. The problem is that they can do this only once or twice. The team members soon will discover that the end date was not really hard and they will loose faith in milestone dates. This is even worse.

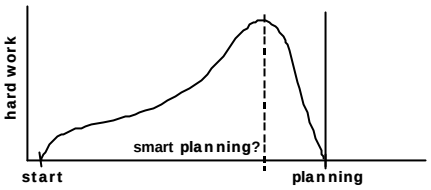


Figure 4: To overcome the late delivery problem, a smart project manager sells his team an earlier delivery date. Even smarter developers soon will know.

The solution for coping with these facts of human psychology is to plan in very short increments (Figure 5). The duration of these increments must be such that:

- The pressure of the end date is felt right the first day.
- The duration of a cycle must be sufficient to finish real tasks.

Three weeks is too long for the pressure and one week may be felt as too short for finishing real tasks. Note that the pressure in this scheme is much healthier than the real stress and failure at the end of a Big Bang (delivery at once at the end) project. The experience in an actual project, where we got only six weeks to finish completely, led to using one-week cycles. The results were such, that we will continue using one-week cycles on all subsequent projects. If you cannot even plan a one-week period, how could you plan longer periods ...?

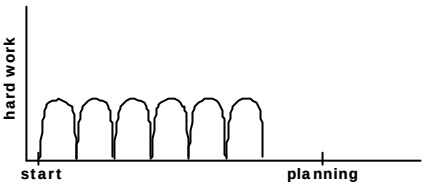


Figure 5: The solution: choose short, realistic "delivery dates". Satisfaction, motivation, fast feedback.

Rapid and frequent feedback

If everything would be completely clear we could use the waterfall development model. We call this production rather than development. At the start of a new development, however, there are many uncertainties we have to explore and to change into certainties. Because even the simplest development project is too complex for a human mind to oversee completely (E. Dijkstra, 1965: "The competent programmer is fully aware of the limited size of his own skull" [12]) we must iteratively learn what we are actually dealing with and learn how to perform better.

This is done by "think first, then do", because thinking costs less than doing. But, because we cannot foresee everything and we have to assume a lot, we constantly have to check whether our thoughts and assumptions were correct. This is called feedback: we plan something, we do it as well as we can, then we check whether the effects are correct. Depending on this analysis, we may change our ways and assumptions. Shewhart already described this in 1939 [13]. Deming [14] called it the Shewhart cycle (Figure 6). Others call it the Deming cycle or PDCA (Plan-Do-Check-Act) cycle.

In practice we see that if developers do something (sec-

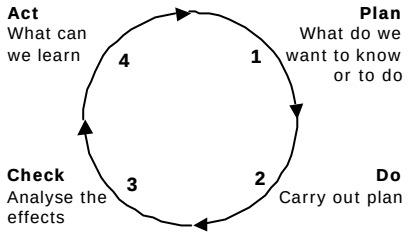


Figure 6: Shewhart cycle, Deming cycle, PDCA cycle.

tion 2 of the cycle), they sometimes plan (section 1), but hardly ever explicitly go through the analysis and learn sections. In Evo we do use all the sections of the cycle deliberately in rapid and frequent feedback loops (Figure 7):

- *The weekly task cycle*

In this cycle we optimise our estimation, planning and tracking abilities in order to better predict the future. We check constantly whether we are *doing* the right things in the right order to the right level of detail for the moment.

- *The frequent stakeholder value delivery cycle*

In this cycle we optimise the requirements and check our assumptions. We check constantly whether we are *delivering* the right things in the right order to the right level of detail for the moment. Delivery cycles may take 1 to 3 weekly cycles.

- *The strategic objectives cycle*

In this cycle we review our strategic objectives and check whether what we do still complies with the objectives. This cycle may take 1 to 3 months.

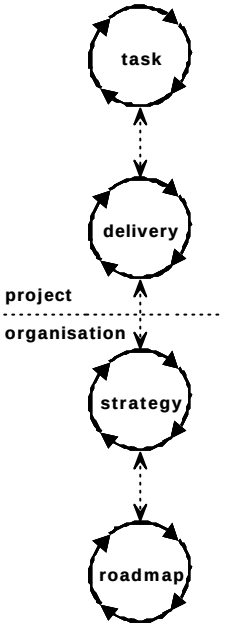


Figure 7: Cycles in Evo.

- *The organisation roadmap cycle*

In this cycle we review our roadmap and check whether our strategic objectives still comply with what we should do in this world. This cycle may take 3 to 6 months.

In development *projects*, only task cycles and delivery cycles are considered. In any task cycle, tasks are done to feed the current delivery, while some other tasks may be done to make future deliveries possible (Figure 8).

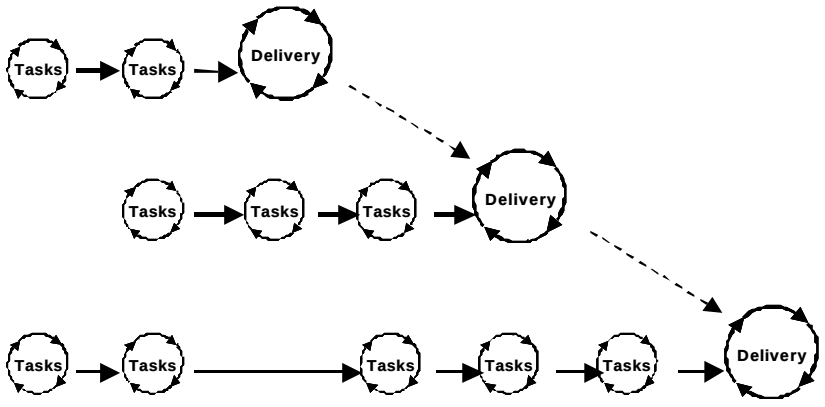


Figure 8: Current tasks feed the current delivery cycle as well as prepare for future delivery cycles.

Time Boxing

Evolutionary project organisation uses *time boxing* rather than *feature boxing.* If we assume that the amount of resources for a given project is fixed, or at least limited, it is possible to realise either:

A fixed set of features in the time needed to realise these features. We call this *feature boxing*.

The amount of features we can realise in a fixed amount of time. We call this *time boxing*.

To realise a fixed set of features in a fixed amount of time with a given set of resources is only possible if the time is sufficient to realise all these features. In practice, however, the time allowed is usually insufficient to realise all the features asked: *What the customer wants, he cannot afford.* If this is the case, we are only fooling ourselves if we try to accomplish the impossible (Figure 9). This has nothing to do with lazy or unwilling developers: if the time (or the budget) is insufficient to realise all the required features, they *will not all be realised*. It is as simple as that.

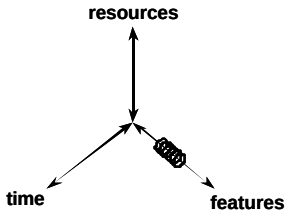


Figure 9: If resources and time are fixed, the features are variable.

The Evo method makes sure that the customer gets the most and most important features *possible* within a certain amount of time and with the available resources. Asking developers to accomplish the impossible is one of the main energy drains in projects. By wasting energy the result is always less than otherwise possible.

In practice, time boxing means:

- A set number of hours is reserved for a task.
- At the end of the time box, the task should be 100% done. That means really *done*.
- Time slip is not allowed in a time box, otherwise other tasks will be delayed and this would lead to uncontrolled delays in the development.
- Before the end of the time box we check how far we can finish the task. If we foresee that we cannot finish a task, we should define what we know now, try to define what we still have to investigate, define tasks and estimate the time still needed. Preferably, however, we should try whether we could *go into less detail* this moment, actually finishing the task to a *sufficient* level of detail within the time box.

  A TaskSheet (details see [8]) is used to define:
  - The goal of the task.
  - The strategy to perform the task.
  - How the result will be verified.
  - How we know for sure that the task is really done (i.e. there is really nothing we have to do any more for this task, we can forget about it).

Estimation, planning and tracking

Estimation, planning and tracking are an inseparable trinity. *If you don't do one of them, you don't* need the other two.

- If you don't estimate, you cannot plan and there is nothing to track.
- If you do not plan, estimation and tracking is useless.
- If you do not track, why should you estimate or plan?

So:

- Derive small tasks from the requirements, the architecture and the overall design.
- Estimate the time needed for every small task.
- Derive the total time needed from:
  - The time needed for all the tasks
  - The available resources
  - Corrected for the real amount of time available per resource (nobody works a full 100% of his presence on the project. The statistical average is about 55%. This is one of the key reasons for late projects! [9])
- Plan the next cycle exactly.
- Be sure that the work of every cycle can be done. That means *really* done. Get commitment from those who are to do the real work.
- Plan the following cycles roughly (the planning may change anyway!).
- Track successes and failures. Learn from it. Refine estimation and planning continuously. Warn stakeholders *well in advance* if the target delivery time is changing because of *any* reason.
- There may be various target delivery times, depending on various feature sets.

If times and dates are not important to you (or to management), then don't estimate, plan, nor track: you don't need it. However, if timing is important, *insist* on estimation, planning and tracking. And it is not even difficult, once you get the hang of it.

If your customer (or your boss) doesn't like to hear that you cannot exactly predict which features will be in at the fatal end day, while you *know* that not all features will be in (at a fixed budget and fixed resources), you can give him two options:

- Either to tell him the day before the fatal day that you did not succeed in implementing all the functions.
- Or tell him now (because you already know), and let him every week decide with you which features are the most important.

It will take some persuasion, but you will see that within two weeks you will work together to get the best possible result. There is one promise you can make: The process used is the most efficient process available. In any other way he will never get more, probably less. So let's work together to make the best of it. Or decide at the beginning to add more resources. Adding resources later evokes Brooks Law [9]: "Adding people to a late project makes it later". Let's stop following ostrich-policy, face reality and deal with it in a realistic and constructive way.

Difference between effort and lead-time

If we ask software developers to estimate a given task in days, they usually come up with estimates of lead-time. If we ask them to estimate a task in hours, they come up with estimates in effort. Project managers know that developers are optimistic and have their private multiplier (like 2, $\sqrt{2}$, e or π) to adjust the estimates given. Because these figures then have to be entered in project-planning tools, like MS Project, they enter the adjusted figures as lead-time.

The problem with lead-time figures is that these are a mix of two different time components:

- Effort, the time needed to do the work
- Lead-time, the time until the work is done. Or rather Lead-time minus Effort, being the time needed for other things than the work to be done. Examples of "other things" are: drinking coffee, meetings, going to the lavatory, discussions, helping colleagues, telephone calls, e-mail, dreaming, etc. In practice we use the Effort/Lead-time ratio, which is usually in the range of 50-70% for full-time team members.

Because the parameters causing variation in these two components are different, they have to be kept apart and treated differently. If we keep planning only in lead-time, we will never be able to learn from the tracking of our planned, estimated figures. Thus we will never learn to predict development time. If these elements are kept separately, people can learn very quickly to adjust their effort estimating intuition. In recent projects we found: first week: 40% of the committed work done, second week: 80% done, from the third week on: 100% or more done. Now we can start predicting!

Separately, people can learn time management to control their Effort/Lead-time ratio. Brooks indicated this already in 1975 [9]: *Programming projects took about twice the expected time. Research showed that half of the time was used for activities other than the project*.

In actual projects, we currently use the rule that people select 2/3 of a cycle (26 hours of 39) for project tasks, and keep 1/3 for other activities. Some managers complain that if we give about 3 days of work and 5 days to do the work, people tend to "Fill the time available". This is called Parkinson's Law [10]: "Work expands so as to fill the time available for its completion". Management uses the same reasoning, giving them 6 days of work and 5 days to do it, hoping to enhance productivity. Because 6 days of effort *cannot* be done in 5 days and people have to do, and *will* do, the other things anyway, people will always *fail to succeed* in accomplishing the impossible. What is worse: this causes a constant sense of failure, causing frustration and demotivation. If we give them the amount of work they can accomplish, they will succeed. This creates a sensation of accomplishment and success, which is very motivating. The observed result is that giving them 3 days work for 5 days is *more productive* that giving them 6 days of work for 5 days.

Commitment

In most projects, when we ask people whether a task is done, they answer: "Yes". If we then ask, "Is it really done?", they answer: "Well, almost". Here we get the effect that if 90% is done, they start working on the other 90%. This is an important cause of delays. Therefore, it is imperative that we define when a task is really 100% done and that we insist that any task be 100% done. Not

100% is *not* done.

In Evo cycles, we ask for tasks to be 100% done. *No need to think about it any more*. Upon estimating and planning the tasks, effort hours have been estimated. Weekly, the priorities are defined. So, every week, when the project manager proposes any team member the tasks for the next cycle, he should never say “Do this and do that”. He should always propose: “Do you still agree that these tasks are highest priority, do you still agree that you should do it, and do you still agree with the estimations?”. If the developer hesitates on any of these questions, the project manager should ask why, and *help the developer* to *re-adjust* such that he can give a *full commitment* that he will accomplish the tasks.

The project manager may help the developer with suggestions (“Last cycle you did not succeed, so maybe you were too optimistic?”). He may *never* take over the responsibility for the decision on which tasks the developer accepts to deliver. This is the only way to get true developer commitment. At the end of the cycle the project manager only has to use the mirror. In the mirror the developer can see himself if he failed in fulfilling his commitments. If the project manager decided what had to be done, the developer sees right through the mirror and only sees the project manager.

It is essential that the project manager coaches the developers in getting their commitments right. Use the sentence: “Promise me to do nothing, as long as *that* is 100% done!” to convey the importance of *completely done*. Only when working with real commitments, developers can learn to optimise their estimations and deliver accordingly. Else, they will *never* learn. Project managers being afraid that the developers will do less than needed and therefore giving the developers more work that they can commit to, will never get what they hope for because without real commitment, people tend to do less.

Risks

If there are no risks whatsoever, use the waterfall model for your development. If there are risks, which is the case in any new development, we have to constantly assess how we are going to control these risks. Development is for an important part risk-reduction. If the development is done, all risks should have been resolved. If a risk turns out for worse at the end of a development, we have no time to resolve it any more. If we identify the risks earlier, we may have time to decide what to do if the risk turns out for worse. Because we develop in very short increments of one week the risk that an assumption or idea consumes a lot of development time before we become aware that the result cannot be used is limited to one week. Every week the requirements are redefined, based upon what we learnt before.

Risks are not limited to assumptions about the product

requirements, where we should ask ourselves:

- Are we developing the right things right?
- When are things right?

Many risks are also about timing and synchronisation:

- Can we estimate sufficiently accurate?
- Which tasks are we forgetting?
- Do we get the deliveries from others (hardware, software, stakeholder responses, …) in time?

Actually the main questions we are asking ourselves systematically in Evo are: *What* should we do, in *which order*, to *which level of detail* for *now*. Too much detail too early means usually that the detail work has to be done over and over again. May be the detail work was not done wrong. It only later turns out that it should have been done differently.

Team meetings

Conventional team meetings usually start with a round of excuses, where everybody tells why he did not succeed in what he was supposed to do. There is a lot of discussion about the work that was supposed to be done, and when the time of the meeting is gone, new tasks are hardly discussed. This is not a big problem, because most participants have to continue their unfinished work anyway. The project manager notes the new target dates of the delayed activities and people continue their work. After the meeting the project manager may calculate how much reserve ("slack time") is left, or how much the project is delayed if all reserve has already been used. In many projects we see that project-planning sheets (MS Project) are mainly used as wallpaper. They are hardly updated and the actual work and the plan-on-the-wall diverge more and more every week.

In the weekly Evo team meeting, we only discuss new work, never past work. We do not waste time for excuses. What is past we cannot change. What we still should do is constantly re-prioritised, so we always work on what is best from this moment. We don't discuss past tasks because they are finished. If discussion starts about the new tasks, we can use the results in our coming work. That can be useful. Still, if the discussion is between only a few participants, it should be postponed till after the meeting, not to waste the others' time.

Magic words

There are several "magic words" that can be used in Evo practice. They can help us to doing the *right things* in the *right order* to the *right level of detail for this moment*.

Focus

Developers tend to be easily distracted by many important or interesting things. Some things may even really be important, however, not at this moment. Keeping focus at the current priority goals, avoiding distractions, is not easy, but saves time.

Priority

Defining priorities and only working on the highest priorities guides us to doing the most important things first.

Synchronise

Every project interfaces with the world outside the project. Active synchronisation is needed to make sure that planned dates can be kept.

Why

This word forces us to define the reason why we should do something, allowing us to check whether it is the right thing to do. It helps in keeping focus.

Dates are sacred

In most projects, dates are fluid. Sacred dates means that if you agree on a date, you stick to your word. Or tell well in advance that you cannot keep your word. With Evo you will know well in advance.

Done

To make estimation, planning and tracking possible, we must finish tasks completely. Not 100% finished is *not* done. This is to overcome the "If 90% is done we continue with the other 90%" syndrome.

Bug, debug

A bug is a small creature, autonomously creeping into your product, causing trouble, and you cannot do anything about it. Wrong. People make mistakes and thus cause defects. The words *bug* and *debug* are dirty words and should be erased from our dictionary. By actively learning from our mistakes, we can learn to avoid many of them. In Evo, we actively catch our mistakes as early as possible and act upon them. Therefore, the impact of the defects caused by our mistakes is minimised and spread through the entire project. This leaves a bare minimum of defects at the end of the project, avoiding the need for a special "debugging phase".

Discipline

With discipline we don't mean imposed discipline, but rather what you, yourself, know what is best to do. If nobody watches us, it is quite human to cut corners, or to do something else, even if we know this is wrong. We see ourselves doing a less optimal thing and we are unable to discipline ourselves. If somebody watches over our shoulder, keeping discipline is easier. So, discipline is difficult, but we can help each other. Evo helps keeping discipline. Why do we want this? Because we enjoy being successful, doing the right things.

How do we use Evo in projects

In our experience, many projects have a mysterious start. Usually when asked to introduce Evo in a project, one or more people have been studying the project already for some weeks or even months. So in most cases, there are some requirements and some idea about the architecture. People acquainted with planning usually already have some idea about what has to be done and have made a conventional planning, based on which the project was proposed and commissioned.

Evo day

To change a project into an Evo project, we organise an "Evo day", typically with the Project Manager, the architect, a tester and *all* other people of the development team. Stakeholder attendance can be useful, but is not absolutely necessary at the first Evo day, where we just teach the team how to change their ways. During the Evo day (and during all subsequent meetings) a notebook and

a LCD projector are used, so that all participants can follow what we are typing and talking about. It is preferable to organise the Evo day outside the company.
The schedule is normally:

Morning
- Presentation of Evo methods [11]: why and how.
- Presentation of the product by the systems architect (people present usually have different views, or even no view, of the product to be developed).

Afternoon

In the afternoon we work towards defining which activities should be worked on in the coming week/cycle. Therefore we do exercises in:

- Defining sub-tasks of max 26 hours.
  In practice, only few activities will be detailed. People get tired of this within 20 minutes, but they did the exercise and anyway we don't have time to do it all in one day.
- Estimating the effort of the sub-tasks, in effort-hours, never in days, see "Difference between effort and lead-time" above.
- Defining priorities.
- Listing the tasks in order of priority.
- Dividing top-priority activities, which have not yet been divided into sub-tasks.
- Estimating effort on top-priority sub-tasks if not yet done.
- The team decides who should do what from the top of the list.
- Every individual developer decides which tasks he will be able to deliver done, really done at the end of the cycle. If a commitment cannot be given, take fewer tasks, until full commitment can be given.

At the end of the day everyone has a list of tasks for the coming week, and a commitment that these tasks will be finished completely, while we are sure that the tasks we start working on have the highest priority.

Last day of the cycle

The last day of a cycle is special and divided into 3 parts (Figure 10):

- The project manager visits every developer individually and discusses the results of the tasks. If the commitments could not be met, they discuss the causes: Was the effort estimation incorrect or was there a time-management problem. The developer should learn from the results to do better the next time. After having visited all developers, the project manager has an overview of the status of the project.
- The status of the project is discussed with the customer, product manager, or whichever relevant stakeholders. Here the Requirements Paradox is handled: during the week, the requirements were fixed, now is the 1 to 2 hours timeslot that the stakeholders may rearrange the requirements and priorities. At the end of this meeting, the requirements and priorities are fixed again.
- Finally, the project manager defines task-proposals for the developers and discusses these proposals with them individually. Developers agree that these tasks have the highest priority and commit to finishing these tasks during the cycle.

Team meeting

Having prepared the individual task-lists for the next cycle, in the team meeting, at the end of the last cycle day, or the beginning of the first new cycle day, the following is done:

- Experience from the past cycle may be discussed if it could benefit subsequent work.
- The status of the project is discussed. Sub-tasks may be (re-)defined and (re-)estimated if full participation is useful.
- The tasks for the next cycle are formally assigned and committed to. Now all participants hear who is going to do what and may react upon it.
- Discussion may be allowed, if it affects most participants.
- The discussions may cause some reprioritisation and thus reshuffling of tasks to be done.

Weekly team meetings typically take less than 20 minutes. A typical reaction at the end of the first Evo team meeting is: "We never before had such a short meeting". When asked "Did we forget to discuss anything important?", the response is: "No, this was a good and efficient meeting". This is one of the ways we are saving time.

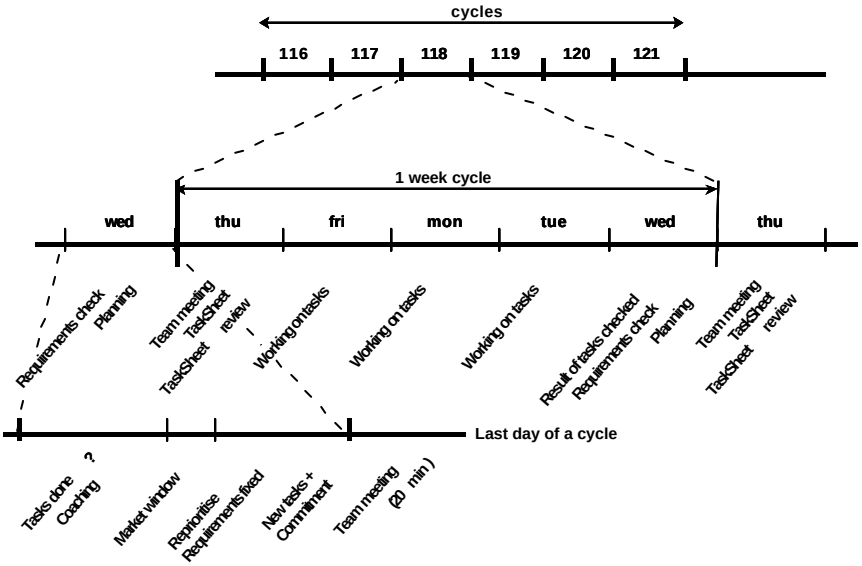


Figure 10: Structure of a weekly cycle

Check lists

There are several checklists being used to help defining priorities and to help to get tasks really finished.

These are currently:

- Task prioritisation criteria
- Delivery prioritisation criteria
- Task conclusion criteria

Task prioritisation criteria

To help in the prioritisation process of which tasks should be done first, we use the following checklist:

- Most important issues first (based on current and future delivery schedules).
- Highest risks first (better early than late).
- Most educational or supporting activities first.
- Synchronisation with the world outside the team (e.g. hardware needs test-software, software needs hardware for test: will it be there when needed?).
- Every task has a useful, completed, working, functional result.

Delivery prioritisation criteria

To help in the prioritisation process of what should be in the next delivery to stakeholders we use the following checklist:

- Every delivery should have the juiciest, most important stakeholder values that can be made in the least time. Impact Estimation [7] is a technique that can be used to decide on what to work on first.
- A delivery must have *symmetrical* stakeholder values. This means that if a program has a start, there must also be an exit. If there is a delete function, there must be also some add function. Generally speaking, the set of values must be a useful whole.
- Every subsequent delivery must show a *clear difference*. Because we want to have stakeholder feedback, the stakeholder must see a difference to feedback on. If the stakeholder feels no difference he feels that he is wasting his time and looses interest to generate feedback in the future.
- Every delivery delivers the *smallest clear increment*. If a delivery is planned, try to delete anything that is not absolutely necessary to fulfil the previous checks. If the resulting delivery takes more than two weeks, try harder.

Task conclusion criteria

If we ask different people about the contents of a defined task, all will tell a more or less different story. In order to make sure that the developer develops the right solution, we use a TaskSheet (details see [8]).
Depending on the task to be done, TaskSheets may be slightly different. First, the developer writes down on the TaskSheet:

- The requirements of the result of the task.
- Which activities must be done to complete the task.
- Design approach: how to implement it.
- Verification approach: how to make sure that it *does* what it *should do* and *does not do* what it should *not do*, based on the requirements.
- Planning (if more than one day work). If this is difficult, ask: “What am I going to do the first day”.
- Anything that is not yet clear.

Then the TaskSheet is reviewed by the system architect. In this process, what the developer *thinks* has to be done is compared with what the system architect *expects*: will the result fit in the big picture? Usually there is some difference between these two views and it is better to find and resolve these differences *before* the actual execution of the task than *after*. This simply saves time.

After agreement, the developer does the work, verifies that the result produced not less, but also *not more*, than the requirements asked for. *Nice things* are not allowed: Anything not specified in the requirements is not tested. Nobody knows about it and this is an irresolvable and therefore unwanted risk.

Finally, the developer uses the task conclusion criteria on the TaskSheet to determine that the task is really done. These criteria may be adapted to certain types of tasks. In practical projects, where software code was written we used the following list:

The code compiles and links with all files in the integration promotion level.

The code simply does what it should do: no bugs.

There are no memory leaks.

A Parkinson test does not make the system crash.

Defensive programming measures have been implemented.

All files are labelled according to the rules agreed.

File promotion is done.

I feel confident that the tester will find no problems.

This checklist is to make sure that the task is really done. If all checks are OK, then the work is done. If it later turns out that the work was not completely done, then the checklist is changed.

Testing with Evo

When developing the conventional way, testing is done at the end of the development, after the Big Bang delivery. Testers then tend to find hundreds of defects, which take a long time to repair. And because there are so many defects, these tend to influence each other. Besides, repairing defects causes more defects.

Software developers are not used to using statistics. If we agree that testing never covers 100% of the software, this means that testing is *taking a sample*. At school we learnt that if we sample, we should use statistics to say something about the whole. So we should get used to statistics and not run away from it.

Statistics tell us that testing is on average 50% effective. Until you have your own (better?) figures, we have to stick to this figure. This means that the user will find the same amount of defects as found in test. Paradoxically this means that the more defects we find in test, the more the user will find. Or, if we do not want the user to find any defects, the test should find no defects *at all*. Most developers think that defect-free software is impossible. If we extrapolate this, it means that we think it is quite normal that our car may stop after a few kilometres drive. Or that the steering wheel in some cases works just the other way: the car turns to the left when we steered to the right… Is that normal?

In Evo, we expect the developers to deliver zero-defect results for the final validation, so that the testers just have to check that everything works OK, as required. Although software developers usually start laughing by this very idea, we are very serious about this. The aim of testing earlier deliveries of Evo cycles is not just testing whether it “works”. Also, testing is not to make life difficult for the developers. In Evo, the software developers ask the testers to help them to find out how far the developers are

from the capability of delivering a defect free product at, or before, final validation (Figure 11).

Figure 11: Testing of early deliveries helps the developers to get ready for zero-defect final delivery.

Change requests and Problem reports

Change Requests (CR) are requested changes in the requirements. Problems Reports (PR) report things found wrong (defects), which we should have done right in the first place. Newly Defined Tasks (NT) are tasks we forgot to define. If any of these is encountered, we never start just changing, repairing, or doing the new task. We work only on defined tasks, of which the effort has been estimated and the priority defined. All tasks are listed on the list of candidate tasks in order of priority. Any CR, PR or NT is first collected in a database. This could be anything between a real database application and a notebook. Regularly, the database is analysed by a Change Control Board (CCB). This could be anything between a very formal group of selected people, who can and must analyse the issues (CRs, PRs and NTs), and an informal group of e.g. the project manager and a team member, who check the database and decide what to do. The CCB can decide to ignore or postpone some issues, to define a new task immediately or to define an analysis task first (Figure 12). In an analysis task, the consequences of the issue are first analysed and an advice is documented about what to do and what the implications are. Any task generated in this process is put on the list of candidate tasks, estimated and prioritised. And only when an existing or new task appears at the top of the candidate list, it will be worked on.

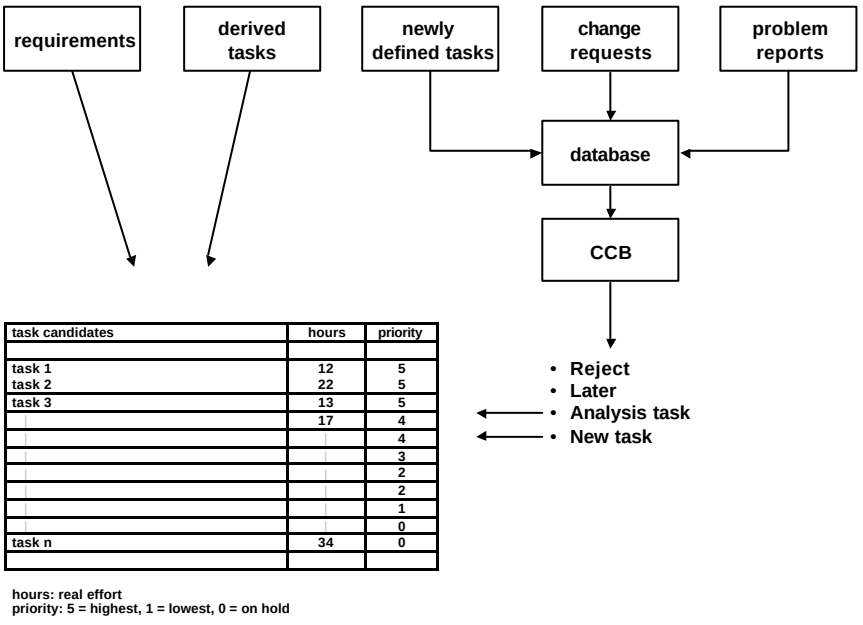


Figure 12: All activities, including Change Requests, Problem Reports and Newly Defined Tasks use the same mechanism for estimation and prioritising: the list of candidate tasks.

Tools

Special tools may only be used when we know and understand the right methods. In actual projects, we have used MS Excel as an easy notepad during interactive sessions with a LCD projector showing what happens on this notepad in real time. When tasks have been defined, MS Project can be used as a spreadsheet to keep track of the tasks per person, while automatically generating a time-line in the Gantt-chart view (Figure 13, top left). This time-line tells people, including management, more than textual planning. It proved possible to let MS Project use weeks of 26 hours and days of 5.2 hours, so that durations could be entered in real effort while the time-line shows correct leadtime-days.

There is a relation between requirements, stakeholder values, deliveries and tasks (Figure 13). We even want to have different views on the list of tasks, like a list of prioritised candidate tasks of the whole project and lists of prioritised tasks per developer. This calls for the use of a relational database, to organise the relations between requirements, values, deliveries and tasks and the different views. Currently, such a database has not been made and the project manager has to keep the consistency of the relations manually. This is some extra work. However, in the beginning it helps the project manager knowing what he is doing. And when we will have found the best way to do it and found the required relationships and views we really need, then we could specify the requirements of a database. If we would have waited till we had a database to keep track of all things, we probably would not have started gaining Evo experience yet. Whether existing tools, like e.g. from Rational can solve this database problem sufficiently, is interesting to investigate.

Important before selecting any tool is, however, to know what we want to accomplish and why and how. Only then we can check whether the tool could save time and burocracy rather that costing time and burocracy.

Conclusion

We described issues that are addressed by the Evo methods and the way we organise Evo projects. By using these methods in actual projects we find:

Faster results

Evo projects deliver better results in 30% shorter time than otherwise. Note: 30% shorter than what by conventional methods would have been achieved. This may be longer than initially hoped for.

Although this 30% is not scientifically proven, it is rather plausible by considering that we constantly check whether we are doing the *right things in the right order to the right level of detail for that moment*. This means that any other process is always less efficient. Most processes (even if you don't know which process you follow, you are following an intuitive *ad hoc* process) cause much

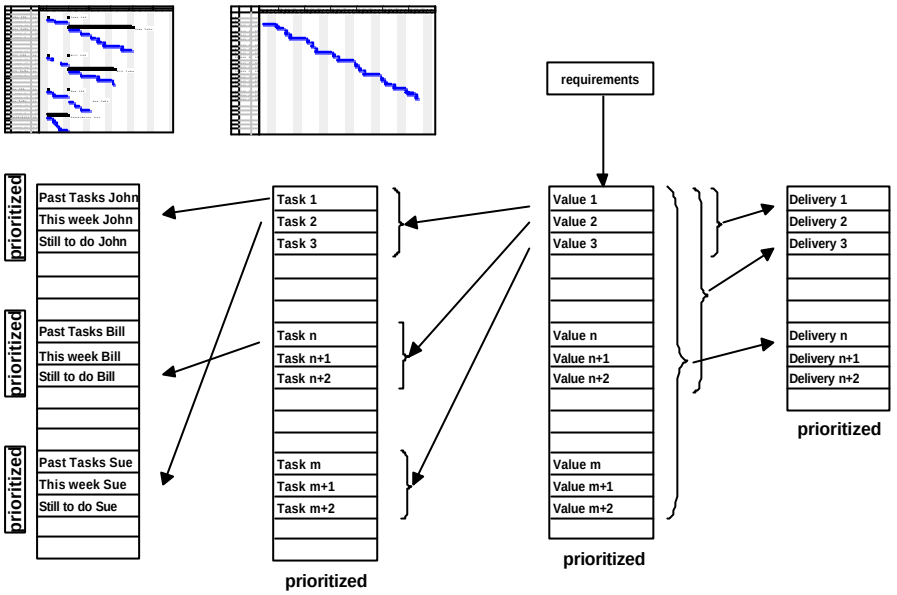


Figure 13: Relations between requirements, stakeholder values, deliveries, and different views on tasks.

work to be done incorrectly and then repaired, as well as unnecessary work. Most developers admit that they use more than half of the total project time on debugging. That is repairing things they did wrong the first time.

Better quality

We define quality as (Crosby [4]) "Conformance to Requirements" (how else can we design for quality and measure quality). In Evo we constantly reconsider the validity of the requirements and our assumptions and make sure that we deliver the most important requirements first. Thus the result will be at least as good as what is delivered with the less rigorous approach we encounter in other approaches.

Less stressed developers

In conventional projects, where it is normal that tasks are not completed in time, developers constantly feel that they fail. This is very demotivating. In Evo projects, developers succeed regularly and see regularly real results of their work. People enjoy success. It motivates greatly. And because motivation is the motor of productivity, the productivity soars. This is what we see happening *within two weeks* in Evo projects: People get relaxed, happy, smiling again, while producing more.

Happy customers

Customers enjoy getting early deliveries and producing regular feedback. They know that they have difficulty in specifying what they really need. By showing them early deliveries and being responsive to their requirements changes, they feel that we know what we are doing. In other developments, they are constantly anxious about the result, which they get only at the end, while experience tells them that the first results are usually not OK and too late. Now they get actual results even much earlier. They start trusting our predictions. And they get a choice of time to market because we deliver complete, functioning results, with growing completeness of functions and qualities, well before the deadline. This has never happened before.

More profits

If we use less time to deliver better quality in a predictable way, we save a lot of money, while we can earn more money with the result. Combined, we make a lot more profit.

In short, although Brooks predicted a long time ago that "There is no silver bullet" [2], we found that the methods presented, which are based on ideas practiced even before the "silver bullet" article, do seem to be a "magic bullet" because of the remarkable results obtained.

Acknowledgement

A lot of the experience with the approach described in this paper has been gained at Philips Remote Control Systems, Leuven, Belgium. In a symbiotic cooperation with the group leader, Bart Vanderbeke, the approach has been introduced in all software projects of his team. Using short discuss-implement-check-act improvement cycles during a period of 8 months, the approach led to a visibly better manageability and an increased comfort-level for the team members. We would like to thank the team members for their contribution to the results.

Niels Malotaux
N R Malotaux – Consultancy
niels@malotaux.nl
www.malotaux.nl/nrm

References

[1] H.D. Mills: Top-Down Programming in Large Systems. In Debugging Techniques in Large Systems. Ed. R. Ruskin, Englewood Cliffs, NJ: Prentice Hall, 1971.

[2] F.P. Brooks, Jr.: No Silver Bullet: essence and Accidents of Software Engineering. In Computer vol 20, no.4 (April 1987): 10-19.

[3] T. Gilb: Principles of Software Engineering Management. Addison-Wesley Pub Co, 1988, ISBN: 0201192462.

[4] P.B. Crosby: Quality Is Still Free. McGraw-Hill, 1996. 4th edition ISBN 0070145326

[5] T. Gilb: manuscript: Evo: The evolutionary Project Managers Handbook. http://www.result-planning.com/pages/2ndLevel/gilbdownload.html, 1997.

[6] S.J.Prowell, C.J.Trammell, R.C.Linger, J.H.Poore: Cleanroom Software Engineering, Technology and process. Addison-Wesley, 1999, ISBN 0201854805.

[7] T. Gilb: manuscript: Impact Estimation Tables: Understanding Complex Technology Quantatively.

http://www.result-planning.com/pages/2ndLevel/gilbdownload.html, 1997.
[8] N.R. Malotaux: TaskSheet. http://www.malotaux.nl/nrm/English/Forms.htm, 2000.
[9] F.P. Brooks, Jr.: The mythical man-month. Addison-Wesley, 1975, ISBN 0201006502. Reprint 1995, ISBN 0201835959.
[10] C. Northcote Parkinson: Parkinsons Law. Buccaneer Books, 1996, ISBN 1568490151.
[11] N.R. Malotaux: Powerpoint slides: Evolutionary Delivery. 2001. http://www.malotaux.nl/nrm/pdf/EvoIntro.pdf.
[12] E. Dijkstra: Paper: Programming Considered as a Human Activity, 1965. Reprint in Classics in Software Engineering. Yourdon Press, 1979, ISBN 0917072146.
[13] W. A. Shewhart: Statistical Method from the Viewpoint of Quality Control. Dover Publications, 1986. ISBN 0486652327.
[14] W.E. Deming: Out of the Crisis. MIT, 1986, ISBN 0911379010.
[15] Kent Beck: Extreme Programming Explained, Addison Wesley, 1999, ISBN 0201616416.
[16] http://www.result-planning.com.
[17] http://www.extremeprogramming.org.

Certificatie van Software Testers

Erik van Veenendaal

Sinds enige tijd bestaan er mogelijkheden om je als tester te laten certificeren. Wederom een teken dat testen een vakgebied is geworden. Enigszins rustig begonnen, is er nu sprake van een ware “hausse” in de testwereld. Er bestaat een tweetal certificeringsprogramma’s, nl. CSTP (Certified Software Testing Professional) in Amerika en het ISEB (Information Systems Examination Board) in de rest van de wereld. Momenteel zijn er wereldwijd reeds meer dan 4.000 testers gecertificeerd op basis van ISEB, waaronder ruim 200 Nederlanders. Het is interessant om te zien dat zowel vanuit de aanbieders als vanuit de leveranciers een grote interesse bestaat voor het certificatietraject. Een ISEB certificaat betekent markterkenning, een maatstaf voor bepaalde bekwaamheden op vele terreinen binnen ICT en wordt ook veelal gekoppeld aan een carrièrestap.

Het ISEB testcertificatie-programma is onderverdeeld in een drietal niveau’s:

1. Foundation Certificate
2. Practitioner Certificate
3. Practitioner Diplome

Het “Foundation Certificate” is reeds een aantal jaren operationeel en bestaat uit een training van drie à vier dagen afgesloten met een examen bestaande uit 40 meerkeuze vragen. Tijdens het “Foundation Certificate” wordt de deelnemer opgeleid en getoetst op het gebied van testprincipes, testfasering, reviews, testtechnieken, testmanagement en testtools. Tot zo’n anderhalf jaar geleden moest men voor deelname aan een foundation training en examen naar het buitenland. Sinds het voorjaar 2000 worden ook cursussen en examens verzorgd in Nederland door Improve Quality Services (driedaagse training), recentelijk zijn ook Polteq en Quality House gestart met de ISEB “Foundation Certificate” training en binnenkort zullen nog één of twee aanbieders aan de vereiste voorwaarden voldoen om de training te mogen verzorgen. Kortom ook in Nederland raakt testcertificatie steeds meer ingeburgerd.

In maart van dit jaar zijn ook de eisen ten aanzien van het “Practitioners Certificate” definitief bekend geworden. Tijdens deze training zal op basis van een omvangrijke case study een testtraject moeten worden doorlopen; er moet een teststrategie worden opgesteld op basis van IEEE 829, testspecificatie en testscripts moeten worden opgesteld op basis van gestructureerde technieken, een testassessment met behulp van TMM moet worden uitgevoerd etc. Het “Practitioners Certificate” omvat een negen- à tiental dagen training, afgesloten met een examen met open vragen. Tijdens het “Practitioners Certificate” komen onderwerpen zoals testmanagement (zo’n drietal dagen), risico-management, technieken (ook non-functional), reviews en inspecties, test process improvement, testtools, en organisatie-aspecten aan de orde. In het examen wordt getoetst op praktische vaardigheden ten aanzien van de genoemde onderwerpen. Momenteel wordt door een aantal trainingsinstituten hard gewerkt aan de totstandkoming van een practitioners cursus die voldoet aan de eisen zoals deze door ISEB worden gesteld. Naar verwachting zullen de eerste trainingen in het tweede kwartaal 2002, ook in Nederland, beschikbaar zijn.

Het zal nog even duren vooraleer het derde niveau “Practitioners Diplome” operationeel is. Voorlopig bieden echter de foundation en practitioner niveaus voldoende uitdagingen. Het ISEB testcertificatietraject biedt de mogelijkheid om een testcarrière in te richten op basis van een extern referentiekader !!

Erik van Veenendaal
eve@improvegs.nl
www.improvegs.nl/certific.html
www.iseb.org.uk

Plenaire sessie: 6 februari 2002, Utrecht

Zalencentrum Bologna, Matthias van Geunsgebouw
Bolognalaan 2, 3584 JK Utrecht

Recent hebben we de release van het CMMI Capability Maturity Model Integrated) gezien. Voor diegene die dat al eens bekeken hebben, zal duidelijk zijn dat er veel veranderd is ten opzicht van het vertrouwde Software CMM. Als belangrijkste veranderingen kunnen we aanmerken de herindeling van KPA's (nu PA's genoemd), de verrijking van de details in de beschrijving van de PA's en de verschillende representaties die het model nu kent (continuous tegenover staged). Aan die laatste verandering willen we de komende plenaire meeting de nodige aandacht besteden.

Aangezien het Software CMM de staged representatie gebruikt, kennen we die natuurlijk allemaal het beste. Al jaren zijn er in de wereld van de modellen-makers sterke voorstanders van een continuous representatie. We hebben voor de komende plenaire meeting een aantal personen gevonden die al een tijd lang bezig zijn met het CMMI en al nagedacht en gediscussieerd hebben over het gebruik van de staged versus de continuous representatie. Op deze avond zullen zij dat met iedereen delen en ook de discussie verder voeren.

Vandaar dat we voor deze avond een andere opzet gekozen hebben:

We beginnen met een inleiding om iedereen vertrouwd te laten worden met de belangrijkste concepten van het CMMI. Daarna zullen Hans Aerts en Ben Linders vertellen wat de voordelen zijn van de verschillende representaties. Hierna mogen de deelnemers zelf in discussie over mogelijke risico's die zij zien. We sluiten dan af met een plenaire discussie op basis van de geïdentificeerde risico's.

We hopen dat met de keuze van het onderwerp en de opzet van de avond er een vruchtbare discussie ontstaat.

Indien u:

- Een presentatie over dit thema wilt verzorgen, en/of
- Zitting wilt nemen in het discusiepanel,

meldt u dan aan bij de activiteitencoördinator van SPIder:
Mark van der Velden,
email: m.van.der.velden@quintgroup.com ,
tel.: 020-305 37 00.

Meer informatie op de website van SPIder:
www.st-spider.nl.

Werkgroepen SPIder

Werkgroep "Testprocesverbetering & SPI"

Onderstaand een korte verslaggeving van de laatste bijeenkomst.

Beste Testprocesverbeteraar,

De bijeenkomst op 20 december was weer geslaagd! We zijn verder gegaan met de vraagstelling zoals we ons de vorige bijeenkomst gesteld hebben: Hoe kan beheerst worden omgegaan met de afstand tussen ontwikkeling en testen? Als eerste stap hebben we een aantal praktijksituaties rond de organisatie van testen besproken (afkomstig uit onze eigen praktijk).

Hiermee gaan we door op woensdagavond 13 februari 2002. Op basis van deze inventarisatie gaan we de omgevingsfactoren benoemen die kenmerkend en identificerend zijn voor elk van de besproken situaties.
De volgende bijeenkomst wordt wederom bij INQA in Vianen gehouden. Ruim 2 weken van te voren ontvang je een defini-tieve uitnodiging.

Contactpersoon:
Dré Robben
mobiel: 06 - 20 777 273
email: robbendr@iquip.nl

Werkgroep "Integrale SPI strategieën"

Onze volgende bijeenkomst vindt plaats op donderdag 14 februari vanaf 16:00 uur (locatie nog niet bekend). De bijeenkomst staat in het teken van "Testen en SPI". Daarnaast heeft de werkgroep dit jaar nog andere interessante onderwerpen op de agenda staan:

- 11 april: Implementatie Prince2
- 13 juni : Implementatie SPI tools
- 12 september: SPI in RAD omgeving
- 14 november: De integrale SPI aanpak

De werkgroep heeft inmiddels de resultaten van een vorige serie van bijeenkomsten op de SPIDER website geplaatst. Mocht je op- of aanmerkingen hebben op de webinformatie, laat het mij dan weten. Zodoende kunnen wij de kwaliteit van dit gedeelte van de website verhogen.

Als je interesse hebt voor de activiteiten van de werkgroep neem dan contact op met Michel Rutgers van SPI Partners.

Contactpersoon: **Michel Rutgers**
tel.: 020-4197211
email: michel.rutgers@spipartners.nl.

Werkgroep "Metrieken"

Contactpersoon: **Hans Vonk**
tel.: 020 - 695 48 57, fax: 020 - 695 27 41
email: Hans@metric.nl

Werkgroep "SPI Invoeringsstrategieën"

De SPIder Werkgroep Invoeringsstrategieen richt zich in ruime zin op alle facetten die te maken hebben met het invoeren van nieuwe werkwijzen. Belangrijke aspecten zijn daarbij het delen van ervaringen en meningen, het bieden van een klankbord voor het bespreken van ideeen en problemen en het volgen van nieuwe ontwikkelingen op het gebied van SPI.

De WG bestaat uit zo'n twintig leden. De leden komen 5 a 6 maal per jaar bijeen op telkens wisselende locaties. De bijeenkomsten beginnen altijd om 16:00 en eindigen rond

20:00 en zijn inclusief een broodjesmaal ter versterking van de inwendige mens.
De bijeenkomsten hebben altijd een onderwerp. De onderwerpen worden door één of meer sprekers Ingeleid, waarna ruimte is voor eigen inbreng en discussie.

De volgende onderwerpen zijn daarbij onder andere aan bod gekomen:

- SPICE/ISO15504
- het inrichten van een verbeterstructuur
- invoering van planning & tracking
- hoe meet je het success van een invoeringsstrategie?
- invoering van quality assurance
- CMMI
- test management
- invoering van metrics

Agenda voor 2002:

- 22 januari, Amsterdam
  Onderwerp: Ervaring met de ondersteuning van een Indiase CMM-L5 organisatie bij de invoering van CMM in NL
- 19 maart, Eindhoven
- 21 mei, Reeuwijk
- 27 augustus, Barneveld
- 15 oktober, Woerden
- 10 december, Eindhoven
  (Onderwerpen volgen)

Sinds kort heeft de WG ook een eigen site, te bereiken onder Werkgroepen via de homepage van SPIder, www.st-spider.nl. Op de website wordt alle relevante informatie (waaronder de gekozen onderwerpen) voor de WG gepubliceerd.
Heb je belangstelling voor deelname aan de WG, meld je dan aan bij de voorzitter, Jarl Meijer. Wellicht tot ziens!

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

Werkgroep "SPI in Kleine Organisaties"

Contactpersonen: **Ger Fischer,** tel. 06 51 357 983
Tjeu Naus, tel: 0495-633221
e-mail: Tjeu.Naus@nbg-industrial.nl

Werkgroep "SPI voor Embedded Software"

Contactpersoon: **Emile van de Logt**
tel.: 040 - 295 77 77, fax: 040 - 295 76 30
email: emile.van.de.logt@cmg.nl

Nieuwsberichten

CMMI SE/SW/IPPD versie 1.1 vrijgegeven

Op 11 januari jl. heeft het SEI versie 1.1 van het CMMI vrijgegeven. Het is te downloaden via de SEI site. Deze versie is, net als versie 1.1 van het SW-CMM, bedoeld om lange tijd stabiel te blijven en als basis te gaan dienen voor toekomstige assessments. De verschillen met versie 1.02 van december 2000 zijn niet heel erg groot.
Een aantal onduidelijkheden is opgelost door betere formuleringen. Wel is in de de process areas Requirements Development, Technical Solution en Product Integration iets meer tekst opgenomen over het ontwikkelen van een product architectuur en designs. Daarnaast zijn de Generic Practices veel duidelijker beschreven.

Het SEI heeft tevens aangekondigd, dat per april 2002 het CMMI nog zal worden uitgebreid met "Acquisition". Het model gaat dan heten: CMMI SE/SW/IPPD/A.

International Software Testing Conference

De 3e Jaarlijkse International Software Testing Conference 2001 heeft plaatsgevonden in Bangalore van 23 november tot 24 november.
Professionals uit het vakgebied stellen hun notities over testmethodieken, uitdagingen, tools, quality control en andere relevante software zaken ter beschikking. Informatie over deze conferentie, verslaglegging van de presentaties door de bijwonende experts e.a. op de site: http://www.softwaredioxide.com/Testing2001.

Introduction to CMM course in UK

This course is a pre-requisite to the CBA IPI Lead Assessor Training, and is fully accredited by CMU-SEI.

The three day training course introduces the CMM five level maturity scale and explains all of the Key Process Areas to provide a complete overview of the concepts of the CMM.

Voor meer informatie over de cursus en/of inschrijving, website http://www.esi.es/Training.CMMengl.pdf, of ESI Training Services, email: training@esi.es
telefoon: 00 34 94 420 9519.

Software & Services Newsletter

Cordis heeft een nieuwsbrief uitgebracht. Deze nieuwsbrief is een aanvulling op de website voor het gebied Technologies and Engineering of Software, Systems en Services.

De nieuwsbrief is in pdf formaat te downloaden via website: http://www.cordis.lu/ist/ka4/tesss.

CMM Series: A practical Approach

De University of California start 4 februari in Santa Clara, California met een serie van 4 individuele cursussen over CMM, te weten:

- Introduction to the CMM: Streamlining the CMM for Today's Oorganizations.
- Focus on the CMM: Implementing the Requirements Management KPA.
- Focus on the CMM: Implementing the Software Project and Tracking Oversight KPAs.
- Focus on the CMM: Implementing the Software Configuration Management KPA.

Meer informatie op website:

http://www.ucsc-extension.edu/main/qd/citlist.taf?-function=detail&start=0&X_Number=X466.

Evenementenkalender

De evenementenkalender bevat een overzicht van internationale conferenties op het gebied van SPI, metrieken en softwareproductkwaliteit. Daarnaast zijn de activiteiten van SPIder opgenomen.

Ook nationale evenementen op het gebied van software-product- en procesverbetering kunnen in deze evenementenkalender worden opgenomen.

Middels de SPIder Koerier kan een organisator van SPI-gerelateerde evenementen een selecte groep van geïnteresseerden bereiken. Voor commerciële evenementen zoals conferenties, workshops, lezingen en andersoortige bijeenkomsten vraagt de redactie een kleine bijdrage in de kosten.

De volgende SPIder Koerier zal volgens planning verschijnen in mei 2002. We verzoeken u om aankondigingen voor de evenementenkalender uiterlijk 14 april bij de redactie te bezorgen. Tussentijdse aankondigingen zijn natuurlijk altijd welkom voor de SPIder website!

Januari

22 januari: **Werkgroep "SPI Invoeringsstrategieën"**, Amsterdam

Onderwerp: Ervaring met de ondersteuning van een indiase CMM-L5 organisatie bij de invoering van CMM in NL.

Contactpersoon: **Jarl Meijer**
telefoon: 06-28.27.3900
email: meijer@dceconsultants.com

29 januari: **Werkgroep "SPI voor Kleine Organisaties"**, Utrecht

Secretariaat: **Tjeu Naus**
telefoon: 0495-633221
email: Tjeu.Naus@nbg-industrial.nl

Februari/Maart

13 februari: **Werkgroep "Testprocesverbetering & SPI"**

Contactpersoon: **Dré Robben**
mobiel: 06 - 20 777 273
email: robbendr@iquip.nl

14 februari: **Werkgroep "Integrale SPI strategieën"**

Contactpersoon: **Michel Rutgers**
tel.: 020-4197211
email: michel.rutgers@spipartners.nl.

februari/maart: **SEPG, Asia Pac**

India:	Tutorials - February 25-28, 2002
	Conference- March 1-2, 2002
Thailand:	Tutorials - March 4-5, 2002
Australia:	Tutorials - March 7-8, 2002
Singapore:	Tutorials - March 18-19, 2002
	Conference- March 20, 2002
China:	Tutorials - March 20-21, 2002
	Conference- March 22, 2002

About the Conference

The Software Engineering Process Group Conference (SEPG) is the leading international conference for software process professionals who champion the improvement of people, process and technology for software excellence in their organis ations.

Concept

To share best practices and papers across Asia Pac and mutually benefit from the rich software experience of Asia Pac academia, industry and government. Several high maturity organisations will get a chance to present their leadership to multiple audience in the region.

Direct all submissions (by mail, fax or e-mail) or for any further inquiry, please contact:

Ms. Amrita C. Sapre, Conference Manager,
Quality Assurance Institute (India) Ltd.,
1013-14A, Ansal Towers, 38, Nehru Place,
New Delhi - 110 019, INDIA.
E-mail: conferences@qaiindia.com
Fax: +91-11-6218974
Tel: +91-11-6220580/6219792
URL: www.qaiindia.com

7 maart: **Werkgroep "SPI voor Kleine Organisaties"**, De Lier

Secretariaat: **Tjeu Naus**
telefoon: 0495-633221
email: Tjeu.Naus@nbg-industrial.nl

18-22 maart: **Software Development Conference Nieuw Zeeland & Australië,** Wellington en Melbourne
Info: http://www.softed.com/sdc2002

19 maart: **Werkgroep "SPI Invoeringsstrategieën"**, Amsterdam

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

April

8 april: **Werkgroep "SPI voor Kleine Organisaties"**

Secretariaat: **Tjeu Naus**
telefoon: 0495-633221
email: Tjeu.Naus@nbg-industrial.nl

9-12 april: **European SEPG 2002, Amsterdam**.
Europa's Software Process Improvement (SPI) Conference.
Info*:* http://www.espi.org/docs/sepgprog.pdf

11 april: **Werkgroep "Integrale SPI strategieën"**

Onderwerp: Implementatie Prince 2

Contactpersoon: **Michel Rutgers**
tel.: 020-4197211
email: michel.rutgers@spipartners.nl.

Mei

21 mei: **Werkgroep "SPI Invoeringsstrategieën**", Amsterdam

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

30 mei: **Werkgroep "SPI voor Kleine Organisaties"**

Secretariaat: **Tjeu Naus**
telefoon: 0495-633221
email: Tjeu.Naus@nbg-industrial.nl

2e helft 2002:

13 juni: **Werkgroep "Integrale SPI strategieën"**
Onderwerp: Implementatie SPI tools

27 augustus: **Werkgroep "SPI Invoeringsstrategieën",** Barneveld

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

12 september: **Werkgroep "Integrale SPI strategieën"**
Onderwerp: SPI in RAD omgeving

15 oktober: **Werkgroep "SPI Invoeringsstrategieën",** Woerden

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

14 november: **Werkgroep "Integrale SPI strategieën"**
Onderwerp: De integrale SPI aanpak

Contactpersoon: **Michel Rutgers**
tel.: 020-4197211
email: michel.rutgers@spipartners.nl.

10 december: **Werkgroep "SPI Invoeringsstrategieën",** Eindhoven

Contactpersoon: **Jarl Meijer**
telefoon:06-28.27.3900
email: meijer@dceconsultants.com

Kijk regelmatig op de SPIder website www.st-spider.nl voor actuele werkgroep mededelingen en bijeenkomsten. De werkgroepen SPI Invoeringsstrategieën, SPI in Kleine Organisaties en Metrieken hebben elk een eigen website, die via www.st-spider.nl is te bereiken.

Colofon

De SPIder redactie bestaat uit:

Renske Henzel, Niels Malotaux, Maarten Wijsman.
Voor reacties en vragen m.b.t. de ***SPIder Koerier*** kunt u zich wenden tot:

Redactie SPIder Koerier, Renske Henzel
Luchthavenweg 81 234, 5657 EA Eindhoven
tel.: 040 - 252 52 92, fax: 040 - 257 21 95
email: rhenzel@visionconsort.nl

Indien u in de toekomst een herinneringsbericht wilt ontvangen over de datum van kopijsluiting, stuur dan een e-mail met "opname SPIder copylijst" naar redactie@st-spider.nl.

Informatie over SPIder is te vinden op de website: www.st-spider.nl.

Voor reacties en bijdragen op de ***SPIder website*** kunt u zich richten tot:

Redactie SPIder web, Niels Malotaux
email: niels@malotaux.nl.

Deze koerier kwam tot stand met medewerking van
- Vision Consort
- N R Malotaux - Consultancy

Deelname in SPIder

Indien u actief wilt participeren in SPIder en de Koerier in de toekomst wilt ontvangen, kunt u zich aanmelden als deelnemer in SPIder bij:

Secretariaat Stichting SPIder
p/a Cantrijn Secretariaten
Postbus 2047, 4200 BA Gorinchem
tel.: 0183 - 62 00 66, fax: 0183 - 62 16 01
email: info@st-spider.nl, website: www.st-spider.nl.

Aanmelding kan ook via het aanmeldingsformulier op de website van SPIder: www.st-spider.nl.